

Parameter Learning in PRISM Programs with Continuous Random Variables

Muhammad Asiful Islam, C.R. Ramakrishnan, I.V. Ramakrishnan

Dept. of Computer Science
Stony Brook University
Stony Brook, NY 11794
{maislam, cram, ram}@cs.sunysb.edu

Abstract

Probabilistic Logic Programming (PLP), exemplified by Sato and Kameya's PRISM, Poole's ICL, De Raedt et al's ProbLog and Vennekens et al's LPAD, combines statistical and logical knowledge representation and inference. Inference in these languages is based on enumerative construction of proofs over logic programs. Consequently, these languages permit very limited use of random variables with continuous distributions. In this paper, we extend PRISM with Gaussian random variables and linear equality constraints, and consider the problem of parameter learning in the extended language. Many statistical models such as finite mixture models and Kalman filter can be encoded in extended PRISM. Our EM-based learning algorithm uses a symbolic inference procedure that represents sets of derivations without enumeration. This permits us to learn the distribution parameters of extended PRISM programs with discrete as well as Gaussian variables. The learning algorithm naturally generalizes the ones used for PRISM and Hybrid Bayesian Networks.

Introduction

Probabilistic Logic Programming (PLP) is a class of Statistical Relational Learning (SRL) frameworks (Getoor and Taskar 2007) which combine statistical and logical knowledge representation and inference. PLP languages, such as SLP (Muggleton 1996), ICL (Poole 2008), PRISM (Sato and Kameya 1997), ProbLog (De Raedt, Kimmig, and Toivonen 2007) and LPAD (Vennekens, Verbaeten, and Bruynooghe 2004) extend traditional logic programming languages by implicitly or explicitly attaching random variables with certain clauses in a logic program. A large class of common statistical models, such as Bayesian networks, Hidden Markov models and Probabilistic Context-Free Grammars have been effectively encoded in PLP; the programming aspect of PLP has also been exploited to succinctly specify complex models, such as discovering links in biological networks (De Raedt, Kimmig, and Toivonen 2007). Parameter learning in these languages is typ-

ically done by variants of the EM algorithm (Dempster, Laird, and Rubin 1977).

Operationally, combined statistical/logical inference in PLP is based on proof structures similar to those created by pure logical inference. As a result, these languages have limited support models with continuous random variables. Recently, we extended PRISM (Sato and Kameya 1997) with Gaussian and Gamma-distributed random variables, and linear equality constraints (<http://arxiv.org/abs/1112.2681>)¹. This extension permits encoding of complex statistical models including Kalman filters and a large class of Hybrid Bayesian Networks.

In this paper, we present an algorithm for parameter learning in PRISM extended with Gaussian random variables. The key aspect of this algorithm is the construction of *symbolic derivations* that succinctly represent large (sometimes infinite) sets of traditional logical derivations. Our learning algorithm represents and computes Expected Sufficient Statistics (ESS) symbolically as well, for Gaussian as well as discrete random variables. Although our technical development is limited to PRISM, the core algorithm can be adapted to parameter learning in (extended versions of) other PLP languages as well.

Related Work SRL frameworks can be broadly classified as statistical-model-based or logic-based, depending on how their semantics is defined. In the first category are languages such as Bayesian Logic Programs (BLPs) (Kersting and De Raedt 2001), Probabilistic Relational Models (PRMs) (Narman et al. 2010), and Markov Logic Networks (MLNs) (Richardson and Domingos 2006), where logical relations are used to specify a model compactly. Although originally defined over discrete random variables, these languages have been extended (e.g. Continuous BLP (Kersting and De Raedt 2001), Hybrid PRM (Narman et al. 2010), and Hybrid MLN (Wang and Domingos 2008)) to support continuous random variables as well. Techniques for parameter learning in statistical-model-based lan-

guages are adapted from the corresponding techniques in the underlying statistical models. For example, discriminative learning techniques are used for parameter learning in MLNs (Singla and Domingos 2005; Lowd and Domingos 2007).

Logic-based SRL languages include the PLP languages mentioned earlier. Hybrid ProbLog (Gutmann, Jaeger, and De Raedt 2010) extends ProbLog by adding continuous probabilistic facts, but restricts their use such that statistical models such as Kalman filters and certain classes of Hybrid Bayesian Networks (with continuous child with continuous parents) cannot be encoded. More recently, Gutmann et al. (2011) introduced a sampling-based approach for (approximate) probabilistic inference in a ProbLog-like language.

Graphical EM (Sato and Kameya 1999) is the parameter learning algorithm used in PRISM. Interestingly, graphical EM reduces to the Baum-Welch (Rabiner 1989) algorithm for HMMs encoded in PRISM. Gutmann et al. (2008) introduced a least squares optimization approach to learn distribution parameters in ProbLog. Co-PrEM (Gutmann, Thon, and De Raedt 2011) is another algorithm for ProLog that computes binary decision diagrams (BDDs) for representing proofs and uses a dynamic programming approach to estimate parameters. BO-EM (Ishihata et al. 2010) is a BDD-based parameter learning algorithm for PRISM. These techniques enumerate derivations (even when represented as BDDs), and do not readily generalize when continuous random variables are introduced.

Background: An Overview of PRISM

PRISM programs have Prolog-like syntax (see Example 1). In a PRISM program the `msw` relation (“multi-valued switch”) has a special meaning: `msw(X, I, V)` says that `V` is a random variable. More precisely, `V` is the outcome of the `I`-th instance from a family `X` of random processes². The set of variables $\{V_i \mid \text{msw}(p, i, V_i)\}$ are i.i.d., and their distribution is given by the random process `p`. The `msw` relation provides the mechanism for using random variables, thereby allowing us to weave together statistical and logical aspects of a model into a single program. The distribution parameters of the random variables are specified separately.

PRISM programs have declarative semantics, called *distribution semantics* (Sato and Kameya 1997; Sato and Kameya 1999). Operationally, query evaluation in PRISM closely follows that for traditional logic programming, with one modification. When the goal selected at a step is of the form `msw(X, I, Y)`, then `Y` is bound to a possible outcome of a random process `X`. The derivation step is associated with the probability of this outcome. If all random processes encountered

²Following PRISM, we often omit the instance number in an `msw` when a program uses only one instance from a family of random processes.

in a derivation are independent, then the probability of the derivation is the product of probabilities of each step in the derivation. If a set of derivations are pairwise mutually exclusive, the probability of the set is the sum of probabilities of each derivation in the set³. Finally, the probability of an answer to a query is computed as the probability of the set of derivations corresponding to that answer.

As an illustration, consider the query `fmix(X)` evaluated over program in Example 1. One step of resolution derives goal of the form `msw(m, M), msw(w(M), X)`. Now depending on the value of `m`, there are two possible next steps: `msw(w(a), X)` and `msw(w(b), X)`. Thus in PRISM, derivations are constructed by enumerating the possible outcomes of each random variable.

Example 1 (Finite Mixture Model) *In the following PRISM program, which encodes a finite mixture model (McLachlan and Peel 2001), `msw(m, M)` chooses one distribution from a finite set of continuous distributions, `msw(w(M), X)` samples `X` from the chosen distribution.*

```
fmix(X) :- msw(m, M),
           msw(w(M), X).

% Ranges of RVs
values(m, [a,b]).
values(w(M), real).
% PDFs and PMFs
:- set_sw(m, [0.3, 0.7]),
   set_sw(w(a), norm(2.0, 1.0)),
   set_sw(w(b), norm(3.0, 1.0)).
```

□

Extended PRISM

Support for continuous variables is added by modifying PRISM’s language in two ways. We use the `msw` relation to sample from discrete as well as continuous distributions. In PRISM, a special relation called `values` is used to specify the ranges of values of random variables; the probability mass functions are specified using `set_sw` directives. We extend the `set_sw` directives to specify probability density functions as well. For instance, `set_sw(r, norm(Mu, Var))` specifies that outcomes of random processes `r` have Gaussian distribution with mean `Mu` and variance `Var`. Parameterized families of random processes may be specified, as long as the parameters are discrete-valued. For instance, `set_sw(w(M), norm(Mu, Var))` specifies a family of random processes, with one for each value of `M`. As in PRISM, `set_sw` directives may be specified programmatically; for instance, the distribution parameters of `w(M)`, may be computed as functions of `M`.

Additionally, we extend PRISM programs with linear equality constraints over reals. Without loss of generality, we assume that constraints are written as linear equalities of the form $Y = a_1 * X_1 + \dots + a_n * X_n + b$ where a_i and b are all floating-point constants. The

³The evaluation procedure is defined only when the independence and exclusiveness assumptions hold.

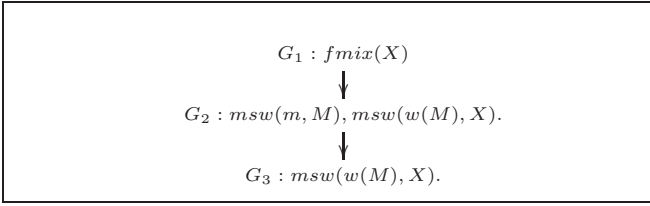


Figure 1: Symbolic derivation for goal $\text{fmix}(X)$

use of constraints enables us to encode Hybrid Bayesian Networks and Kalman Filters as extended PRISM programs. In the following, we use *Constr* to denote a set (conjunction) of linear equality constraints. We also denote by \overline{X} a vector of variables and/or values, explicitly specifying the size only when it is not clear from the context. This permits us to write linear equality constraints compactly (e.g., $Y = \overline{a} \cdot \overline{X} + b$).

Inference

The key to inference in the presence of continuous random variables is avoiding enumeration by representing the derivations and their attributes symbolically. A single step in the construction of a symbolic derivation is defined below.

Definition 1 (Symbolic Derivation) A goal G directly derives goal G' , denoted $G \rightarrow G'$, if:

PCR: $G = q_1(\overline{X}_1), G_1$, and there exists a clause in the program, $q_1(\overline{Y}) : r_1(\overline{Y}_1), r_2(\overline{Y}_2), \dots, r_m(\overline{Y}_m)$, such that $\theta = \text{mgu}(q_1(\overline{X}_1), q_1(\overline{Y}))$; then, $G' = (r_1(\overline{Y}_1), r_2(\overline{Y}_2), \dots, r_m(\overline{Y}_m), G_1)\theta$;

MSW: $G = \text{msw}(\text{rv}(\overline{X}), Y), G_1$; then $G' = G_1$;

CONS: $G = \text{Constr}, G_1$ and *Constr* is satisfiable; then $G' = G_1$.

A symbolic derivation of G is a sequence of goals G_0, G_1, \dots such that $G = G_0$ and, for all $i \geq 0$, $G_i \rightarrow G_{i+1}$.

Note that the traditional notion of derivation in a logic program coincides with that of symbolic derivation when the selected subgoal (literal) is not an **msw** or a constraint. When the selected subgoal is an **msw**, PRISM's inference will construct the next step by enumerating the values of the random variable. In contrast, symbolic derivation skips **msw**'s and constraints and continues with the remaining subgoals in a goal. The effect of these constructs is computed by associating (a) variable type information and (b) a success function (defined below) with each goal in the derivation. The symbolic derivation for the goal **fmix**(X) over the program in Example 1 is shown in Fig. 1.

Success Functions: Goals in a symbolic derivation may contain variables whose values are determined by **msw**'s appearing subsequently in the derivation. With each goal G_i in a symbolic derivation, we associate a set of variables, $V(G_i)$, that is a subset of variables in G_i . The set $V(G_i)$ is such that the variables in $V(G_i)$ subsequently appear as parameters or outcomes of **msw**'s in

some subsequent goal G_j , $j \geq i$. We can further partition V into two disjoint sets, V_c and V_d , representing continuous and discrete variables, respectively.

Given a goal G_i in a symbolic derivation, we can associate with it a *success function*, which is a function from the set of all valuations of $V(G_i)$ to $[0, 1]$. Intuitively, the success function represents the probability that the symbolic derivation represents a successful derivation for each valuation of $V(G_i)$. Note that the success function computation uses a set of distribution parameters Θ . For simplicity, we often omit it in the equations and use it when it's not clear from the context.

Representation of success functions: Given a set of variables \mathbf{V} , let \mathbf{C} denote the set of all linear equality constraints over reals using \mathbf{V} . Let \mathbf{L} be the set of all linear functions over \mathbf{V} with real coefficients. Let $\mathcal{N}_X(\mu, \sigma^2)$ be the PDF of a univariate Gaussian distribution with mean μ and variance σ^2 , and $\delta_x(X)$ be the Dirac delta function which is zero everywhere except at x and integration of the delta function over its entire range is 1. Expressions of the form $k * \prod_l \delta_{v_l}(V_l) \prod_i \mathcal{N}_{f_i}$, where k is a non-negative real number and $f_i \in \mathbf{L}$, are called *product PDF (PPDF) functions over \mathbf{V}* . We use ϕ (possibly subscripted) to denote such functions. A pair $\langle \phi, C \rangle$ where $C \subseteq \mathbf{C}$ is called a *constrained PPDF function*. A sum of a finite number of constrained PPDF functions is called a *success function*, represented as $\sum_i \langle \phi_i, C_i \rangle$.

We use $C_i(\psi)$ to denote the constraints (i.e., C_i) in the i^{th} constrained PPDF function of success function ψ ; and $D_i(\psi)$ to denote the i^{th} PPDF function of ψ .

Success functions of base predicates: The success function of a constraint C is $\langle 1, C \rangle$. The success function of *true* is $\langle 1, \text{true} \rangle$. The PPDF component of $\text{msw}(\text{rv}(\overline{X}), Y)$'s success function is the probability density function of rv 's distribution if rv is continuous, and its probability mass function if rv is discrete; its constraint component is *true*.

Example 2 The success function of $\text{msw}(m, M)$ for the program in Example 1 is $\psi_1 = 0.3\delta_a(M) + 0.7\delta_b(M)$.

The success function $\text{msw}(w(M), X)$ for the program in Example 1 is $\psi_2 = \delta_a(M)\mathcal{N}_X(2.0, 1.0) + \delta_b(M)\mathcal{N}_X(3.0, 1.0)$. \square

Success functions of user-defined predicates: If $G \rightarrow G'$ is a step in a derivation, then the success function of G is computed bottom-up based on the success function of G' . This computation is done using *join* and *marginalize* operations on success functions.

Definition 2 (Join) Let $\psi_1 = \sum_i \langle D_i, C_i \rangle$ and $\psi_2 = \sum_j \langle D_j, C_j \rangle$ be two success functions, then *join* of ψ_1 and ψ_2 represented as $\psi_1 * \psi_2$ is the success function $\sum_{i,j} \langle D_i D_j, C_i \wedge C_j \rangle$.

Given a success function ψ for a goal G , the success function for $\exists X. G$ is computed by the marginalization operation. Marginalization w.r.t. a discrete

variable is straightforward and omitted. Below we define marginalization w.r.t. continuous variables in two steps: first rewriting the success function in a projected form and then doing the required integration.

Projection eliminates any linear constraint on V , where V is the continuous variable to marginalize over. The projection operation, denoted by $\psi \downarrow_V$, involves finding a linear constraint (i.e., $V = \bar{a} \cdot \bar{X} + b$) on V and replacing all occurrences of V in the success function by $\bar{a} \cdot \bar{X} + b$.

Proposition 1 *Integration of a PPDF function with respect to a variable V is a PPDF function, i.e.,*

$$\begin{aligned} \alpha \int_{-\infty}^{\infty} \prod_{k=1}^m \mathcal{N}_{(\bar{a}_k \cdot \bar{X}_k + b_k)}(\mu_k, \sigma_k^2) dV \\ = \alpha' \prod_{l=1}^{m'} \mathcal{N}_{(\bar{a}'_l \cdot \bar{X}'_l + b'_l)}(\mu'_l, \sigma'^2_l) \end{aligned}$$

where $V \in \bar{X}_k$ and $V \notin \bar{X}'_l$.

Definition 3 (Integration) *Let ψ be a success function that does not contain any linear constraints on V . Then integration of ψ with respect to V , denoted by $\oint_V \psi$ is a success function ψ' such that $\forall i. D_i(\psi') = \int D_i(\psi) dV$.*

Definition 4 (Marginalize) *Marginalization of a success function ψ with respect to a variable V , denoted by $\mathbb{M}(\psi, V)$, is a success function ψ' such that*

$$\psi' = \oint_V \psi \downarrow_V$$

We overload \mathbb{M} to denote marginalization over a set of variables, defined such that $\mathbb{M}(\psi, \{V\} \cup \bar{X}) = \mathbb{M}(\mathbb{M}(\psi, V), \bar{X})$ and $\mathbb{M}(\psi, \{\}) = \psi$.

The success function for a derivation is defined as follows.

Definition 5 (Success function of a derivation) *Let $G \rightarrow G'$. Then the success function of G , denoted by ψ_G , is computed from that of G' , based on the way G' was derived:*

PCR: $\psi_G = \mathbb{M}(\psi_{G'}, V(G') - V(G))$.

MSW: Let $G = \text{msw}(\text{rv}(\bar{X}), Y), G_1$. Then $\psi_G = \psi_{\text{msw}(\text{rv}(\bar{X}), Y)} * \psi_{G'}$.

CONS: Let $G = \text{Constr}, G_1$. Then $\psi_G = \psi_{\text{Constr}} * \psi_{G'}$.

Note that the above definition carries PRISM's assumption that an instance of a random variable occurs at most once in any derivation. In particular, the PCR step marginalizes success functions w.r.t. a set of variables; the valuations of the set of variables must be mutually exclusive for correctness of this step. The MSW step joins success functions; the goals joined must use independent random variables for the join operation to correctly compute success functions in this step.

Example 3 *Fig. 1 shows the symbolic derivation for goal $\text{fmix}(X)$ over the finite mixture model program in Example 1. Success function of goal G_3 is $\psi_{\text{msw}(w(M), X)}(M, X)$, hence $\psi_{G_3} = \delta_a(M) \mathcal{N}_X(\mu_a, \sigma_a^2) + \delta_b(M) \mathcal{N}_X(\mu_b, \sigma_b^2)$.*

ψ_{G_2} is $\psi_{\text{msw}(m, M)}(M) * \psi_{G_3}(M, X)$ which yields $\psi_{G_2} = p_a \delta_a(M) \mathcal{N}_X(\mu_a, \sigma_a^2) + p_b \delta_b(M) \mathcal{N}_X(\mu_b, \sigma_b^2)$. Note that $\delta_b(M) \delta_a(M) = 0$ as M can not be both a and b at the same time. Also $\delta_a(M) \delta_a(M) = \delta_a(M)$.

Finally, $\psi_{G_1} = \mathbb{M}(\psi_{G_2}, M)$ which is $p_a \mathcal{N}_X(\mu_a, \sigma_a^2) + p_b \mathcal{N}_X(\mu_b, \sigma_b^2)$. Note that ψ_{G_1} represents the mixture distribution (McLachlan and Peel 2001) of mixture of two Gaussian distributions.

Here $p_a = 0.3, p_b = 0.7, \mu_a = 2.0, \mu_b = 3.0$, and $\sigma_a^2 = \sigma_b^2 = 1.0$. \square

Note that for a program with only discrete random variables, there may be exponentially fewer symbolic derivations than concrete derivations *à la* PRISM. The compactness is only in terms of *number* of derivations and not the total size of the representations. In fact, for programs with only discrete random variables, there is a one-to-one correspondence between the entries in the tabular representation of success functions and PRISM's answer tables. For such programs, it is easy to show that the time complexity of the inference presented in this paper is same as that of PRISM.

Learning

We use the expectation-maximization algorithm (Dempster, Laird, and Rubin 1977) to learn the distribution parameters from data. First we show how to compute the expected sufficient statistics (ESS) of the random variables and then describe our algorithm.

The ESS of a discrete random variable is a n -tuple where n is the number of values that the discrete variable takes. Suppose that a discrete random variable V takes v_1, v_2, \dots, v_n as values. Then the ESS of V is $(ESS^{V=v_1}, ESS^{V=v_2}, \dots, ESS^{V=v_n})$ where $ESS^{V=v_i}$ is the expected number of times variable V had valuation v_i in all possible proofs for a goal. The ESS of a Gaussian random variable X is a triple $(ESS^{X, \mu}, ESS^{X, \sigma^2}, ESS^{X, \text{count}})$ where the components denote the expected sum, expected sum of squares and the expected number of uses of random variable X , respectively, in all possible proofs of a goal. When derivations are enumerated, the ESS for each random variable can be represented by a tuple of reals. To accommodate *symbolic* derivations, we lift each component of ESS to a function, represented as described below.

Representation of ESS functions: For each component ν (discrete variable valuation, mean, variance, total counts) of a random variable, its ESS function in a goal G is represented as follows:

$$\xi_G^\nu = \sum_i \langle \chi_i \phi_i, C_i \rangle.$$

where $\langle \phi_i, C_i \rangle$ is a constrained PPDF function and

$$\chi_i = \begin{cases} \bar{a}_i \cdot \bar{X}_i + b_i & \text{if } \nu = X, \mu \\ \bar{a}_i \cdot \bar{X}_i^2 + b_i & \text{if } \nu = X, \sigma^2 \\ b_i & \text{otherwise} \end{cases}$$

Here \bar{a}_i, b_i are constants, and $\bar{X}_i = V_c(G)$.

Note that the representation of ESS function is same as that of success function for discrete random variable valuations and total counts. *Join* and *Marginalize* operations, defined earlier for success functions, can be readily defined for ESS functions as well. The computation of ESS functions for a goal, based on the symbolic derivation, uses the extended *join* and *marginalize* operations. The set of all ESS functions is closed under the extended *Join* and *Marginalize* operations.

ESS functions of base predicates: The ESS function of the i^{th} parameter of a discrete random variable V is $P(V = v_i)\delta_{v_i}(V)$. The ESS function of the mean of a continuous random variable X is $X\mathcal{N}_X(\mu, \sigma^2)$, and the ESS function of the variance of a continuous random variable X is $X^2\mathcal{N}_X(\mu, \sigma^2)$. Finally, the ESS function of the total count of a continuous random variable X is $\mathcal{N}_X(\mu, \sigma^2)$.

Example 4 In this example, we compute the ESS functions of the random variables (m , $w(a)$, and $w(b)$) in Example 1. According to the definition of ESS function of base predicates, the ESS functions of these random variables for goals $msw(m, M)$ and $msw(w(M), X)$ are

ESS	for $msw(m, M)$	for $msw(w(M), X)$
ξ^k	$p_k\delta_k(M)$	0
ξ^{μ_k}	0	$X\delta_k(M)\mathcal{N}_X(\mu_k, \sigma_k^2)$
$\xi^{\sigma_k^2}$	0	$X^2\delta_k(M)\mathcal{N}_X(\mu_k, \sigma_k^2)$
ξ^{count_k}	0	$\delta_k(M)\mathcal{N}_X(\mu_k, \sigma_k^2)$

where $k \in \{a, b\}$. \square

ESS functions of user-defined predicates: If $G \rightarrow G'$ is a step in a derivation, then the ESS function of a random variable for G is computed bottom-up based on the its ESS function for G' .

The ESS function of a random variable component in a derivation is defined as follows.

Definition 6 (ESS functions in a derivation) Let $G \rightarrow G'$. Then the ESS function of a random variable component ν in the goal G , denoted by ξ_G^ν , is computed from that of G' , based on the way G' was derived:

PCR: $\xi_G^\nu = \mathbb{M}(\xi_{G'}^\nu, V(G') - V(G))$.

MSW: Let $G = msw(rv(\bar{X}), Y), G_1$. Then $\xi_G^\nu = \psi_{msw(rv(\bar{X}), Y)} * \xi_{G'}^\nu + \psi_{G'} * \xi_{msw}^\nu$.

CONS: Let $G = Constr, G_1$. Then $\xi_G^\nu = \psi_{Constr} * \xi_{G'}^\nu$.

Example 5 Using the definition of ESS function of a derivation involving MSW, we compute the ESS function of the random variables in goal G_2 of Fig. 1.

	ESS functions for goal G_2
ξ^k	$p_k\delta_k(M)\mathcal{N}_X(\mu_k, \sigma_k^2)$
ξ^{μ_k}	$Xp_k\delta_k(M)\mathcal{N}_X(\mu_k, \sigma_k^2)$
$\xi^{\sigma_k^2}$	$X^2p_k\delta_k(M)\mathcal{N}_X(\mu_k, \sigma_k^2)$
ξ^{count_k}	$p_k\delta_k(M)\mathcal{N}_X(\mu_k, \sigma_k^2)$

Notice the way $\xi_{G_2}^k$ is computed.

$$\begin{aligned} \xi_{G_2}^k &= \psi_{msw(m, M)}\xi_{G_3}^k + \psi_{G_3}\xi_{msw(m, M)}^k \\ &= [p_a\delta_a(M) + p_b\delta_b(M)] \cdot 0 \\ &\quad + [\delta_a(M)\mathcal{N}_X(\mu_a, \sigma_a^2) + \delta_b(M)\mathcal{N}_X(\mu_b, \sigma_b^2)] \cdot p_k\delta_k(M) \\ &= p_k\delta_k(M)\mathcal{N}_X(\mu_k, \sigma_k^2) \end{aligned}$$

Finally, for goal G_1 we marginalize the ESS functions w.r.t. M .

	ESS functions for goal G_1
ξ^k	$p_k\mathcal{N}_X(\mu_k, \sigma_k^2)$
ξ^{μ_k}	$Xp_k\mathcal{N}_X(\mu_k, \sigma_k^2)$
$\xi^{\sigma_k^2}$	$X^2p_k\mathcal{N}_X(\mu_k, \sigma_k^2)$
ξ^{count_k}	$p_k\mathcal{N}_X(\mu_k, \sigma_k^2)$

\square

The algorithm for learning distribution parameters (Θ) uses a fixed set of training examples (t_1, t_2, \dots, t_N). Note that the success and ESS functions for t_i 's are constants as the training examples are variable free (i.e., all the variables get marginalized over).

Algorithm 1 (Expectation-Maximization)

Initialize the distribution parameters Θ .

1. Construct the symbolic derivations for ψ and ξ using current Θ .
2. **E-step:** For each training example t_i ($1 \leq i \leq N$), compute the ESS (ξ_{t_i}) of the random variables, and success probabilities ψ_{t_i} w.r.t. Θ .

M-step: Compute the MLE of the distribution parameters given the ESS and success probabilities (i.e., evaluate Θ'). Θ' contains updated distribution parameters (p', μ', σ'^2). More specifically, for a discrete random variable V , its parameters are updated as follows:

$$p'_{V=v} = \frac{\eta_{V=v}}{\sum_{u \in \text{values}(V)} \eta_{V=u}}$$

where

$$\eta_{V=v} = \sum_{i=1}^N \frac{\xi_{t_i}^{V=v}}{\psi_{t_i}}.$$

For each continuous random variable X , its mean and variances are updated as follows:

$$\mu'_X = \frac{\sum_{i=1}^N \frac{\xi_{t_i}^{X, \mu}}{\psi_{t_i}}}{N_X}$$

$$\sigma'^2_X = \frac{\sum_{i=1}^N \frac{\xi_{t_i}^{X, \sigma^2}}{\psi_{t_i}}}{N_X} - \mu'^2_X$$

where N_X is the expected total count of X .

$$N_X = \sum_{i=1}^N \frac{\xi_{t_i}^{X, \text{count}}}{\psi_{t_i}}$$

3. Evaluate the log likelihood ($\ln P(t_1, \dots, t_N | \Theta') = \sum_i \ln \psi_{t_i}$) and check for convergence. Otherwise let $\Theta \leftarrow \Theta'$ and return to step 1.

Theorem 2 Algorithm 1 correctly computes the MLE which (locally) maximizes the likelihood.

(Proof) Sketch. The main routine of Algorithm 1 for discrete case is same as the *learn-naive* algorithm of Sato and Kameya (1999), except the computation of $\eta_{V=v}$.

$$\eta_{V=v} = \sum_{\text{for each goal } g} \frac{1}{\psi_g} \sum_S P(S) N_S^v.$$

where S is an explanation for goal g and N_S^v is the total number of times $V = v$ in S .

We show that $\xi_g^{V=v} = \sum_S P(S) N_S^v$.

Let the goal g has a single explanation S where S is a conjunction of subgoals (i.e., $S_{1:n} = g_1, g_2, \dots, g_n$). Thus we need to show that $\xi_g^{V=v} = P(S) N_S^v$.

We prove this by induction on the length of S . The definition of ξ for base predicates gives the desired result for $n = 1$. Let the above equation holds for length n i.e., $\xi_{g_{1:n}}^{V=v} = P(S_{1:n}) N_{1:n}^v$. For $S_{1:n+1} = g_1, g_2, \dots, g_n, g_{n+1}$,

$$\begin{aligned} P(S_{1:n+1}) N_{1:n+1}^v &= P(g_1, g_2, \dots, g_n, g_{n+1}) N_{1:n+1}^v \\ &= P(g_1, g_2, \dots, g_n) P(g_{n+1}) (N_{1:n}^v + N_{n+1}^v) \\ &= P(S_{1:n}) P(g_{n+1}) N_{1:n}^v + P(S_{1:n}) P(g_{n+1}) N_{n+1}^v \\ &= P(g_{n+1}) [P(S_{1:n}) N_{1:n}^v] + P(S_{1:n}) [P(g_{n+1}) N_{n+1}^v] \\ &= P(g_{n+1}) \xi_{g_{1:n}}^{V=v} + P(S_{1:n}) \xi_{g_{n+1}}^{V=v} \\ &= \xi_{g_{1:n+1}}^{V=v} \end{aligned}$$

The last step follows from the definition of ξ in a derivation.

Now based on the exclusiveness assumption, for disjunction (or multiple explanations) like $g = g_1 \vee g_2$ it trivially follows that $\xi_g^{V=v} = \xi_{g_1}^{V=v} + \xi_{g_2}^{V=v}$.

Example 6 Let x_1, x_2, \dots, x_N be the observations. For a given training example $t_i = \text{fmix}(x_i)$, the ESS functions are

	ESS functions for goal $\text{fmix}(x_i)$
ξ_k	$p_k \mathcal{N}_X(x_i \mu_k, \sigma_k^2)$
ξ_{μ_k}	$x_i p_k \mathcal{N}_X(x_i \mu_k, \sigma_k^2)$
$\xi_{\sigma_k^2}$	$x_i^2 p_k \mathcal{N}_X(x_i \mu_k, \sigma_k^2)$
ξ_{count_k}	$p_k \mathcal{N}_X(x_i \mu_k, \sigma_k^2)$

The E-step of the EM algorithm involves computation of the above ESS functions.

In the M-step, we update the model parameters from the computed ESS functions.

$$p'_k = \frac{\sum_{i=1}^N \frac{\xi_{t_i}^k}{\psi_{t_i}}}{\sum_{i=1}^N \frac{\xi_{t_i}^a}{\psi_{t_i}} + \frac{\xi_{t_i}^b}{\psi_{t_i}}} = \frac{\sum_{i=1}^N \frac{\xi_{t_i}^k}{\psi_{t_i}}}{N} \quad (1)$$

Similarly,

$$\mu'_k = \frac{\sum_{i=1}^N \frac{\xi_{t_i}^{\mu_k}}{\psi_{t_i}}}{N_k} \quad (2)$$

$$\sigma'^2_k = \frac{\sum_{i=1}^N \frac{\xi_{t_i}^{\sigma_k^2}}{\psi_{t_i}}}{N_k} - \mu'^2_k \quad (3)$$

where $k \in \{a, b\}$ and

$$N_k = \sum_{i=1}^N \frac{\xi_{t_i}^{\text{count}_k}}{\psi_{t_i}} \quad (4)$$

Example 7 This example illustrates that for the mixture model example, our ESS computation does the same computation as standard EM learning algorithm for mixture models (Bishop 2006).

Notice that for Equation 1, $\frac{\xi_{t_i}^k}{\psi_{t_i}} = \frac{p_k \mathcal{N}_k(x_i | \mu_k, \sigma_k^2)}{\sum_l p_l \mathcal{N}_l(x_i | \mu_l, \sigma_l^2)}$ which is nothing but the posterior responsibilities presented in (Bishop 2006).

$$p'_k = \frac{\sum_{i=1}^N \frac{\xi_{t_i}^k}{\psi_{t_i}}}{N} = \frac{\sum_{i=1}^N \frac{p_k \mathcal{N}_k(x_i | \mu_k, \sigma_k^2)}{\sum_l p_l \mathcal{N}_l(x_i | \mu_l, \sigma_l^2)}}{N}$$

Similarly for Equation 2,

$$\mu'_k = \frac{\sum_{i=1}^N \frac{\xi_{t_i}^{\mu_k}}{\psi_{t_i}}}{N_k} = \frac{\sum_{i=1}^N \frac{x_i p_k \mathcal{N}_k(x_i | \mu_k, \sigma_k^2)}{\sum_l p_l \mathcal{N}_l(x_i | \mu_l, \sigma_l^2)}}{\sum_{i=1}^N \frac{p_k \mathcal{N}_k(x_i | \mu_k, \sigma_k^2)}{\sum_l p_l \mathcal{N}_l(x_i | \mu_l, \sigma_l^2)}}.$$

Variances are updated similarly. \square

Discussion and Concluding Remarks

The symbolic inference and learning procedures enable us to reason over a large class of statistical models such as hybrid Bayesian networks with discrete child-discrete parent, continuous child-discrete parent (finite mixture model), and continuous child-continuous parent (Kalman filter), which was hitherto not possible in PLP frameworks. It can also be used for hybrid models, e.g., models that mix discrete and Gaussian distributions. For instance, consider the mixture model example (Example 1) where $\mathbf{w}(\mathbf{a})$ is Gaussian but $\mathbf{w}(\mathbf{b})$ is a discrete distribution with values 1 and 2 with 0.5 probability each. The density of the mixture distribution can be written as

$$f(X) = 0.3 \mathcal{N}_X(2.0, 1.0) + 0.35 \delta_{1.0}(X) + 0.35 \delta_{2.0}(X)$$

Thus the language can be used to model problems that lie outside traditional hybrid Bayesian networks.

ProbLog and LPAD do not impose PRISM's mutual exclusion and independence restrictions. Their inference technique first materializes the set of explanations for each query, and represents this set as a BDD, where each node in the BDD is a (discrete) random variable. Distinct paths in the BDD are mutually exclusive and

variables in a single path are all independent. Probabilities of query answers are computed trivially based on this BDD representation. The technical development in this paper is limited to PRISM and imposes its restrictions. However, by materializing the set of symbolic derivations first, representing them in a factored form (such as a BDD) and then computing success functions on this representation, we can readily lift the restrictions for the parameter learning technique.

This paper considered only univariate Gaussian distributions. Traditional parameter learning techniques have been described for multivariate distributions without introducing additional machinery. Extending our learning algorithm to the multivariate case is a topic of future work.

References

- [Bishop 2006] Bishop, C. 2006. *Pattern recognition and Machine Learning*. Springer.
- [De Raedt, Kimmig, and Toivonen 2007] De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A probabilistic prolog and its application in link discovery. In *IJCAI*, 2462–2467.
- [Dempster, Laird, and Rubin 1977] Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B* 39(1):1–38.
- [Getoor and Taskar 2007] Getoor, L., and Taskar, B. 2007. *Introduction to Statistical Relational Learning*. The MIT Press.
- [Gutmann et al. 2008] Gutmann, B.; Kimmig, A.; Kersting, K.; and De Raedt, L. 2008. Parameter learning in probabilistic databases: A least squares approach. In *ECML/PKDD (1)*, 473–488.
- [Gutmann et al. 2011] Gutmann, B.; Thon, I.; Kimmig, A.; Bruynooghe, M.; and De Raedt, L. 2011. The magic of logical inference in probabilistic programming. *TPLP* 11(4-5):663–680.
- [Gutmann, Jaeger, and De Raedt 2010] Gutmann, B.; Jaeger, M.; and De Raedt, L. 2010. Extending ProbLog with continuous distributions. In *Proceedings of ILP*.
- [Gutmann, Thon, and De Raedt 2011] Gutmann, B.; Thon, I.; and De Raedt, L. 2011. Learning the parameters of probabilistic logic programs from interpretations. In *ECML/PKDD (1)*, 581–596.
- [Ishihata et al. 2010] Ishihata, M.; Kameya, Y.; Sato, T.; and ichi Minato, S. 2010. An em algorithm on bdds with order encoding for logic-based probabilistic models. *Journal of Machine Learning Research - Proceedings Track* 13:161–176.
- [Kersting and De Raedt 2001] Kersting, K., and De Raedt, L. 2001. Adaptive Bayesian logic programs. In *Proceedings of ILP*.
- [Lowd and Domingos 2007] Lowd, D., and Domingos, P. 2007. Efficient weight learning for markov logic networks. In *PKDD*, 200–211.
- [McLachlan and Peel 2001] McLachlan, G., and Peel, D. 2001. *Finite mixture models*. Wiley.
- [Muggleton 1996] Muggleton, S. 1996. Stochastic logic programs. In *Advances in inductive logic programming*.
- [Narman et al. 2010] Narman, P.; Buschle, M.; König, J.; and Johnson, P. 2010. Hybrid probabilistic relational models for system quality analysis. In *Proceedings of EDOC*.
- [Poole 2008] Poole, D. 2008. The independent choice logic and beyond. In *Probabilistic ILP*, 222–243.
- [Rabiner 1989] Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, 257–286.
- [Richardson and Domingos 2006] Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning*.
- [Sato and Kameya 1997] Sato, T., and Kameya, Y. 1997. PRISM: a symbolic-statistical modeling language. In *IJCAI*.
- [Sato and Kameya 1999] Sato, T., and Kameya, Y. 1999. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research* 391–454.
- [Singla and Domingos 2005] Singla, P., and Domingos, P. 2005. Discriminative training of markov logic networks. In *AAAI*, 868–873.
- [Vennekens, Verbaeten, and Bruynooghe 2004] Vennekens, J.; Verbaeten, S.; and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *ICLP*, 431–445.
- [Wang and Domingos 2008] Wang, J., and Domingos, P. 2008. Hybrid markov logic networks. In *Proceedings of AAAI*.